

# Introduction to Data Science

**BRIAN D'ALESSANDRO**  
**ADJUNCT PROFESSOR, NYU**  
**FALL 2018**

*Fine Print: these slides are, and always will be a work in progress. The material presented herein is original, inspired, or borrowed from others' work. Where possible, attribution and acknowledgement will be made to content's original source. Do not distribute, except for as needed as a pedagogical tool in the subject of Data Science.*

# LETS BUILD A DATASET FOR SPAM FILTERING



# THE BEAUTY OF SPAM

Fighting email SPAM is an incredibly interesting DS problem, as it encompasses many elements of production grade DS challenges.

- Problem is highly adversarial, so constant updating is necessary
- Problem has high potential for selection bias and negative feedback loops
- Problem is inherently multi-task
- Features are mixed text, categorical and numeric
- Data is very high dimensional, sparse and large

# SPAM – TARGET & SAMPLING

1. What is the Target Variable?
2. What does it look like?
3. How do we know if an email is spam or not?
4. What is the likely distribution of the observed target variable? How should we sample it?

# SPAM – FEATURES

Lets look at an example spam and identify aspects that give it away.  
How should we encode these as features?

READ CAREFULLY AND KEEP CONFIDENTIAL



Spam x



P



mr rifat muhammad rifat.muhammad101@gmail.com via yahoo.com

1:10 PM (3 hours ago) ☆



n



1

**⚠ Be careful with this message.** Many people marked similar messages as phishing scams, so this might contain unsafe content. [Learn more](#)

Good Day My Good Friend,

How are you doing together with your entire family, I hope all is well? Please carefully read and understand my reason of contacting you through this email. I am Monsieur Mr Rifat Muhammad. The Chief International Relationship Manager Foreign Remittance Unit Ecobank (ECB). Ouagadougou Burkina - Faso West Africa.

I want to contact you for a very beneficial business transaction. There is an abandoned sum of Twelve Million Five Hundred Thousand United State Dollars. (\$12.5 Million) that was deposited by a late customer of this bank called Mr. Paul Wellstone.

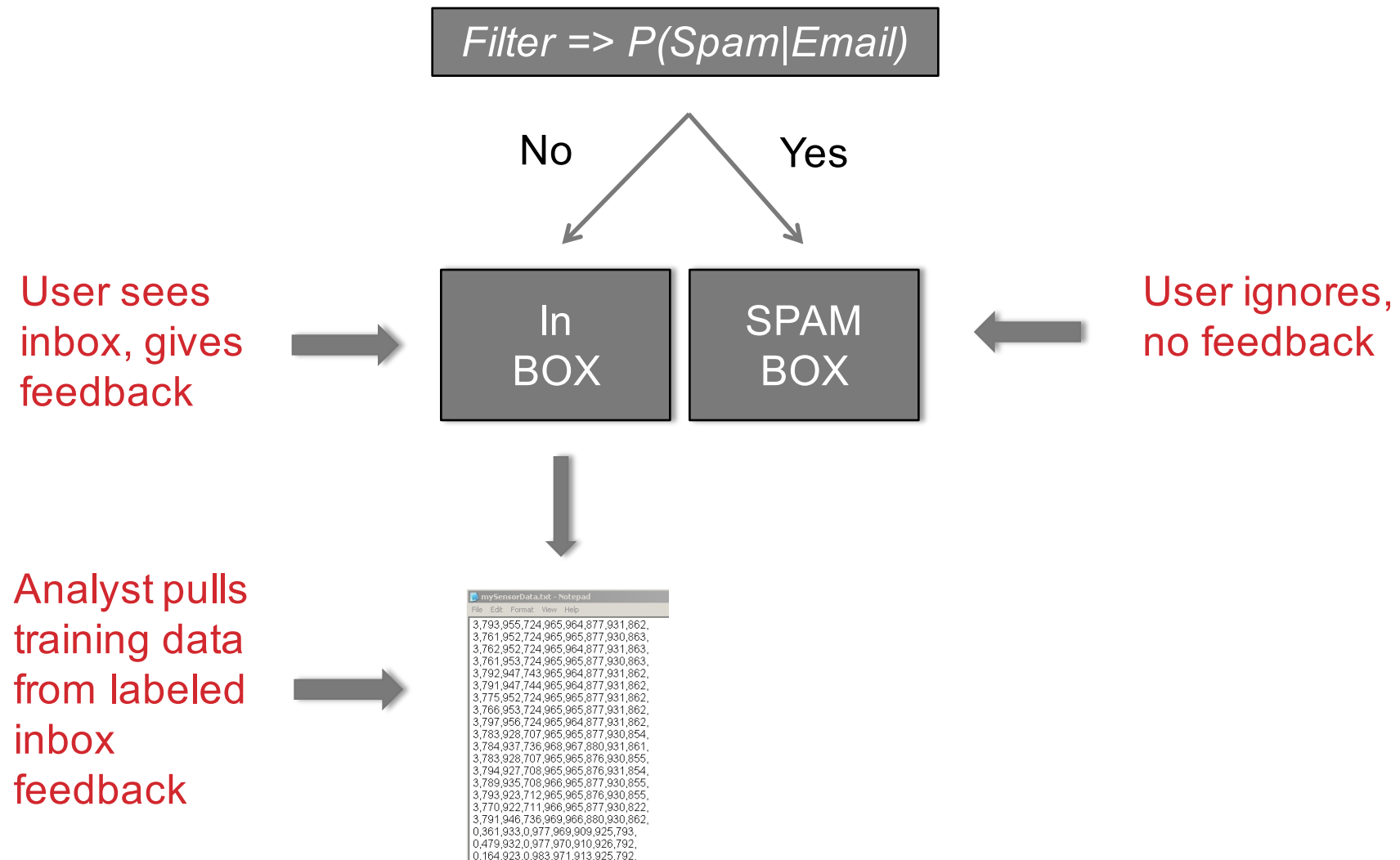
A very prominent man who was doing business transaction with my bank here in Burkina Faso. He was a citizen of United state of America and It's was after my research that i realized about his death.

The bank has no knowledge about his death up to date. Upon this discovery that i decided to make this business proposal to you since the banking laws and guidelines here stipulate that if such funds remain unclaimed here after 12 years years, the money will be transferred into the Central Bank Treasury as unclaimed fund.

# SPAM – FEEDBACK LOOP

Congrats, you are ready to update your SPAM filter!

Now lets anticipate what could go wrong, including negative feedback loops.



# SPAM – THOUGHT STARTERS

1. If the training data is pulled only from labeled Inbox, is selection bias likely? Why?
2. Should you put some SPAM in the user's inbox so that you can get more training labels?
3. What if user deletes w/o reading but doesn't label spam?
4. How often should we retrain the model?
5. What would happen if you completely retrained model from scratch but didn't include examples from the SPAM box?

# SPAM – THOUGHT STARTERS

1. If the training data is pulled only from labeled Inbox, is selection bias likely? Why? **Yes.** Features that highly predict SPAM will be less represented in the inbox. This could have big implications on what future training iterations learn.
2. Should you put some SPAM in the user's inbox so that you can get more training labels? This is the best way to get an unbiased dataset, but too much SPAM makes the service less usable. Best to apply 'active learning' techniques. Can also sample more obvious cases from SPAM box to include in training data.
3. What if user deletes w/o reading but doesn't label spam? This is tricky. Best approach might be to train models with and without these entries labeled as SPAM, and AB test them.
4. How often should we retrain the model? This is an empirical question. If SPAMmers get smarter, more SPAM will end up in Inbox. We can monitor SPAM labeling rates and retrain when they start to trend upwards.
5. What would happen if you completely retrained model from scratch but didn't include examples from the SPAM box? Features that were highly predictive of SPAM, but not observed in Inbox may not get learned in the new model. Thus you end up with a negative feedback loop. One could use the techniques discussed above, or also build the model in an incremental, online fashion. That way learned predictive features that are no longer observed won't change.



# TEXT FEATURE EXTRACTION

# RAW TEXT

*This is how text usually comes...*

Doc	Phrase
1	the cow jumped over the moon
2	somewhere over the rainbow
3	over the moon

*Though this isn't exactly useful in ML applications.*

# TOKENIZING

A very straightforward and scalable way to deal with text is to convert individual words to tokens, and let each token be a separate feature in the data.

## *Original Data Representation*

Doc	Phrase
1	the cow jumped over the moon
2	somewhere over the rainbow
3	over the moon

## *Tokenized Data Representation (Binary Representation)*

Doc	FID						
	1	2	3	4	5	6	7
1	1	1	1	1	0	0	1
2	0	0	0	1	1	1	1
3	0	0	1	1	0	0	1

## *Feature Dictionary*

Token	FID
cow	1
jumped	2
moon	3
over	4
rainbow	5
somewh ere	6
the	7

# TOKENIZING - PYTHON

[http://scikit-learn.org/stable/modules/feature\\_extraction.html](http://scikit-learn.org/stable/modules/feature_extraction.html)

```
>>> from sklearn.feature_extraction.text import CountVectorizer
```

```
>>> vectorizer = CountVectorizer(min_df=1)
```

1. Instantiate an object of type **CountVectorizer**

```
>>> vectorizer
```

```
CountVectorizer(analyzer=... 'word', binary=False,
                 charset_error=None, decode_error=... 'strict',
                 dtype=<... 'numpy.int64'>, encoding=... 'utf-8', input=... 'content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                 strip_accents=None, token_pattern=... '(?u)\b\w+\b',
                 tokenizer=None, vocabulary=None)
```

```
>>> corpus = [
...     'This is the first document.',
...     'This is the second second document.',
...     'And the third one.',
...     'Is this the first document?',
... ]
```

```
>>> X = vectorizer.fit_transform(corpus)
```

```
>>> X
```

```
<4x9 sparse matrix of type '<... 'numpy.int64'>'
with 19 stored elements in Compressed Sparse ... format>
```

2. Fit a sequence of texts

3. Returns data in a sparse matrix format.

# SPARSE FORMAT

By default, `sklearn.feature_extraction.text` returns matrices in “sparse” format. In many applications, especially text, there are many features but each instance only has a relatively few non-zero values. We can define a format that does not explicitly store zeros to save space.

*Dense Format*

FID							
Doc	1	2	3	4	5	6	7
1	1	1	1	1	0	0	1
2	0	0	0	1	1	1	1
3	0	0	1	1	0	0	1



*Sparse Format  
(Explicit Values)*


Doc	Sparse Features
1	1:1 2:1 3:1 4:1 7:1
2	4:1 5:1 6:1 7:1
3	3:1 4:1 7:1

*Sparse Format  
(Implicit Values)*

Doc	Sparse Features
1	1 2 3 4 7
2	4 5 6 7
3	3 4 7

# EXTENTIONS - NGRAMS

To gain more flexibility in our feature representation, we might want to tokenize combinations of words, as opposed to the words themselves. Often times word combinations can express more nuance, such as when negative sentiment is being expressed (i.e., “I don’t like”)

*“I like rainbows.  
They look nice.”*  *[ (“I like”), (“like rainbows”), ...,  
 (“They look”), (“look nice”) ]*

Each n-gram can then be considered an individual feature.

To build n-grams in sklearn:

```
>>> vect = CountVectorizer(min_df=1, analyzer='char', ngram_range=(1,2))  
>>> X = vect.fit_transform(corpus)
```

# EXTENTIONS – TF/IDF

In many cases we want more than binary indicators that a particular word/ngram is present in the document. One popular method, TF/IDF, assigns a heuristic importance weight of each word/ngram to each document.

$$TF - IDF = Term\_Freq \times Inverse\_Document\_Freq$$

Term frequency measures how many times a particular token appears in a particular document, and gives an indication of how important that word is to that document.

$$TF(term, Doc) = \sum_{word \in Doc} \mathbb{I}(word == term)$$

Inverse document frequency measures how often a word appears across all documents. It produces a measure that penalizes words that appear frequently.

$$IDF(term, Corpus) = \log \frac{|Corpus|}{\sum_{Doc \in Corp} \mathbb{I}(term \in Doc)}$$

# TF/IDF - SKLEARN

TF-IDF can be implemented directly with tokenization using the `sklearn.feature_extraction.text.TfidfVectorizer` class.

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer(min_df=1)
>>> vectorizer.fit_transform(corpus)
...
<4x9 sparse matrix of type '<... 'numpy.float64'>'
  with 19 stored elements in Compressed Sparse ... format>
```

## Variations:

- TF-IDF row vectors can be rescaled to have unit norms
- TF can be calculating using a binary indicator of (freq>0) or log(freq)
- IDF can be augmented to prevent division by zero

**The optimality of TF-IDF is of course an empirical question.  
Testing is always recommended!**



# NAÏVE BAYES

# BAYES RULE

In classification we want to estimate:  $P(y|x_1, x_2, \dots, x_m)$

If we remember Bayes rule:  $\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$

So mathematically:

$$P(y|x_1, x_2, \dots, x_m) = \frac{P(y)P(x_1, x_2, \dots, x_m|y)}{P(x_1, x_2, \dots, x_m)}$$

**How many parameters must we estimate for the above model?**

# THE NAÏVE PART

We make one assumption that simplifies our estimation problem tremendously. This is that each feature is independent of each other, condition on  $Y=y$ . Mathematically, this translates to:

$$P(x_i|y, x_j) = P(x_i|y)$$

We now apply this conditional independence assumption to the joint distribution of  $X$  given  $Y=y$ .

$$P(x_1, x_2, \dots, x_m|y) = P(x_1|y)P(x_2|y) \dots P(x_m|y) = \prod_{i=1}^m P(x_i|y)$$

**Why do we make this assumption?**

# CONSTRUCTING A CLASSIFIER

We start with the fact that  $P(x_1, x_2, \dots, x_m)$  is constant given the data (and also not dependent on the class value).

Which leads us to this relation for each value of  $Y=y$ :

$$P(y|x_1, x_2, \dots, x_m) \propto P(y) \prod_{i=1}^m P(x_i|y)$$

We then add a decision rule, which is to choose the value of  $y$  that is the most probable. This leads to the following maximum a posteriori decision rule:

$$\hat{y} = \operatorname{argmax}_{y \in Y} P(y) \prod_{i=1}^m P(x_i|y)$$

# NB AS A LINEAR MODEL

We can position NB as another type of linear model. Assuming balanced classes, the previous optimization problem can be expressed as:

$$\hat{y} = \mathbb{I}\left(\frac{P(y=1|x_1, x_2, \dots, x_m)}{P(y=0|x_1, x_2, \dots, x_m)} > 1\right)$$

And using Bayes rule again, the log of the quantity in the above indicator function can be written as:

$$f(x) = \ln \frac{P(y=1)}{P(y=0)} + \sum_{i=1}^m \ln \frac{P(x_i|y=1)}{P(x_i|y=0)} = \alpha + \beta \cdot x$$

Which makes our NB estimator:

$$\hat{y} = \mathbb{I}(f(x) > 0)$$

# VARIATIONS OF NB

## Multinomial Naïve Bayes

In MNB we assume each document is characterized by a set of words/tokens (here called  $x_i$ , and that each word/token has a weight (usually frequency of occurrence or tf-idf for the document).

The posterior distribution is given by 
$$P(y|doc) \propto P(y)P(doc|y) \propto p(y) \prod_{x_i \in doc} P(x_i|y)^{freq_i}$$

$P(x_i|y)$  is a smoothed maximum likelihood estimate, where  $N_{yi}$  is the total document count of a word given  $y$ ,  $N_y$  is the document count of all words given  $y$ ,  $\alpha$  is a smoothing parameter (usually=1),  $n$  is count of all distinct words.

$$P(x_i|y) = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

We can use this to define a decision rule:

$$\hat{y} = \operatorname{argmax}_{y \in Y} \log[P(y)P(doc|y)] = \operatorname{argmax}_{y \in Y} [\log(P(y)) + \sum_{x_i \in doc} w_i \log(\frac{N_{yi} + \alpha}{N_y + \alpha n})]$$

With the weight  $w_i$  being either the frequency or the tf-idf score of the word/token in the given document.

**References:** [http://scikit-learn.org/stable/modules/naive\\_bayes.html](http://scikit-learn.org/stable/modules/naive_bayes.html),

“Tackling the Poor Assumptions of Naive Bayes Text Classifiers”, Rennie et al, ICML-2003

# VARIATIONS OF NB

## Bernoulli Naïve Bayes

In BNB, each word/token  $x_i$  is a binary indicator, where  $w_i$  indicates that  $x_i$  is in a given document. The posterior distribution is then given by:

$$P(y|doc) \propto P(y)P(doc|y) = p(y) \prod_i P(x_i|y)^{w_i} (1 - P(x_i|y))^{1-w_i}$$

In this scenario we explicitly account for the fact that a given  $x_i$  is not in a particular document, and  $P(x_i|y)$  is again the smoothed maximum likelihood estimate that word  $x_i$  appears in a given document given  $y$ . I.e., # of documents containing  $x_i$  in the class over the # of documents in the class.

Our decision function is then.

$$\hat{y} = \operatorname{argmax}_{y \in Y} [\log(P(y)) + \sum_i w_i \log(P(x_i)) + (1 - w_i) \log(1 - P(x_i))]$$

**References:** [http://scikit-learn.org/stable/modules/naive\\_bayes.html](http://scikit-learn.org/stable/modules/naive_bayes.html),  
“Naive Bayes and Text Classification I”, Sebastian Raschka

# VARIATIONS OF NB

## Gaussian Naïve Bayes

GNB is the standard formulation for continuously valued features. Again we use the following decision rule:

$$\hat{y} = \operatorname{argmax}_{y \in Y} P(y) \prod_{i=1}^m P(x_i|y)$$

We make the assumption that  $X_i$  is distributed as a normal random variable:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_{iy}^2}} \exp\left(-\frac{(x_i - \mu_{iy})^2}{\sigma_{iy}^2}\right)$$

Where:

$$\mu_{iy} = E[X_i|y] \qquad \sigma_{iy}^2 = E[(X_i - \mu_{iy})^2|y]$$

**References:** <http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf>

Generative and Discriminative Classifiers: Naïve Bayes and Logistic Regression



# NOTES/ADVANTAGES

## **Naïve Bayes fast and scalable**

- estimates each parameter separately
- can handle sparse data and is easily parallelized
- No numeric optimization – just counting (easy in a query language)

## **Good classifier, bad as a generative model**

- if Naïve assumption holds, NB is a Bayes optimal classifier
- Naïve assumption never holds, so  $P(Y|X)$  tends to be biased towards 0 or 1
- Nonetheless, works well under 0/1 loss

## **Sparsity Can Hurt You**

- $P(x|y)$  needs to be smoothed, usually with beta-prior on the binomial distribution (LaPlace smoothing)

## **Class Imbalance**

- Binary decision rule works best when classes are roughly equal
- Will always pick dominant class in practice if high skew exists
- Can use posterior estimate of  $P(Y|X)$  (or log) to rank and then use ROC to define optimal cut-off point.